PBD Assignment 01

Md Mobashir Rahman and Adwitiya Argha Priyadarshini Boruah

November 04, 2025

Author: Md Mobashir Rahman and Adwitiya Argha Priyadarshini Boruah
Email: mdra00001@stud.uni-saarland.de, adbo00002@stud.uni-saarland.de
Matriculation Number: 7059086, 7070291

1 Principal Component Analysis (PCA) and Data Imputation

1.1 Exercise: Principal Component Analysis (40 points)

Write a program that applies the PCA technique to the toy dataset in the supplement ("pca toy.txt").

-> (a): Standardize the variables in the dataset. Why is this step necessary before performing a PCA?

1.1.1 (a) Standardize the variables before running PCA:

We first load the toy dataset and standardize each variable to have mean 0 and standard deviation 1. Standardization is necessary so that variables measured on different scales contribute equally to the principal components; otherwise, variables with larger variances would dominate the PCA solution.

```
pca_raw <- read.delim("Assignment_1_supplement/pca_toy.txt", check.names = FALSE)
pca_scaled <- scale(pca_raw)
pca_summary <- data.frame(
  variable = colnames(pca_raw),
  mean = apply(pca_scaled, 2, mean),
  sd = apply(pca_scaled, 2, sd)
)
pca_summary
```

```
##   variable          mean sd
## a        a -4.950207e-16  1
## b        b  1.920686e-16  1
## c        c -1.143530e-16  1
## d        d  3.486100e-15  1
```

Observation: Column means are ~0 and SDs are 1 (tiny numerical noise is expected).

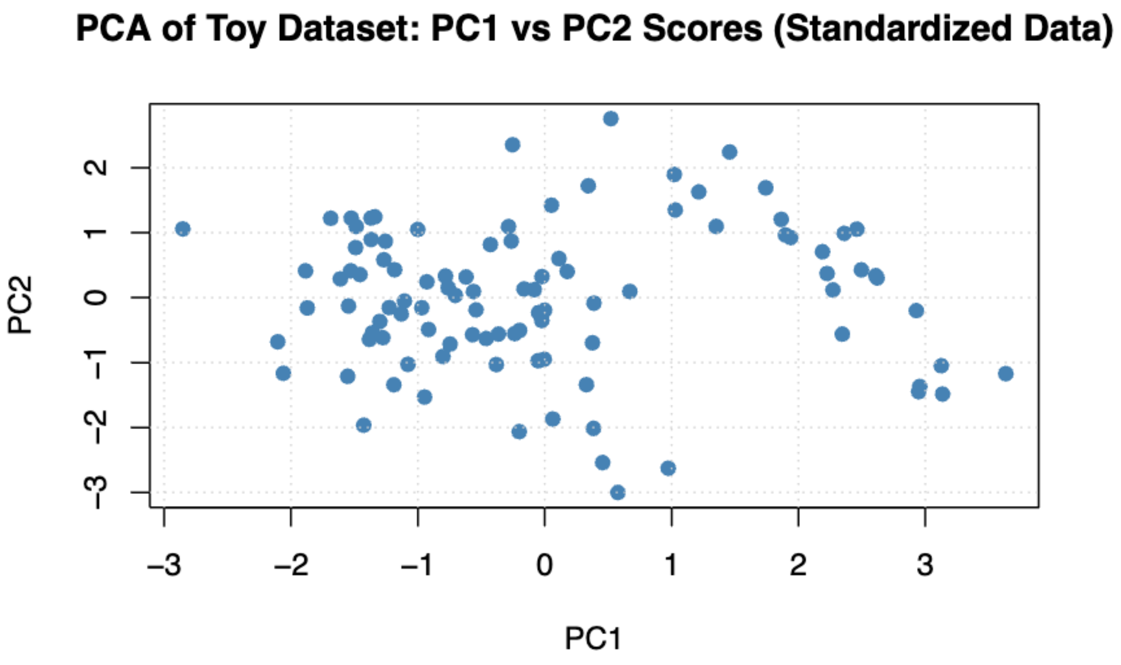
-> (b): Create a scatter plot from the transformed data, with PC1 and PC2 as the axes.

1.1.2 (b) Run PCA and create a scatter plot of PC1 vs PC2:

We compute the PCA on standardized variables and plot the scores on PC1 vs PC2.

```
pca_model <- prcomp(pca_scaled, center = FALSE, scale. = FALSE)
pca_scores <- as.data.frame(pca_model$x[, 1:2])
```

```
plot(
  pca_scores$PC1,
  pca_scores$PC2,
  xlab = "PC1",
  ylab = "PC2",
  main = "PCA of Toy Dataset: PC1 vs PC2 Scores (Standardized Data)",
  pch = 19,
  col = "steelblue"
)
grid()
```



-> (c): Which variables are the most important for PC1 and PC2? Why?

1.1.3 (c) Identify the most important variables for PC1 and PC2:

We examine the loadings (i.e., the coefficients of the eigenvectors) for PC1 and PC2. Variables with larger absolute loadings contribute more to the corresponding component.

```r
pca_loadings <- pca_model$rotation[, 1:2]
pca_loadings
```

```
##          PC1        PC2
## a -0.4357230  0.5053838
## b -0.6113755 -0.3355008
## c  0.2745604 -0.7132847
## d -0.6008179 -0.3510717
```

From the loadings we can identify the variables with the largest absolute values for each component. PC1 is most influenced by b and d, while PC2 is most influenced by c and a. These variables have the highest leverage in shaping the corresponding components.

-> (d) :What percentage of the variance in the dataset is explained by PC1 and PC2?

1.1.4 (d) Determine the variance explained by PC1 and PC2:

We compute the proportion of variance explained by each principal component and report the first two.

```r
importance_table <- summary(pca_model)$importance
importance_table
```

```
##                            PC1      PC2       PC3       PC4
## Standard deviation     1.47734 1.116475 0.6980192 0.2893467
## Proportion of Variance 0.54563 0.311630 0.1218100 0.0209300
## Cumulative Proportion  0.54563 0.857260 0.9790700 1.0000000
```

PC1 and PC2 together explain 85.73% of the variance in the standardized dataset.

-> (e): Under what circumstances would more PCs be considered?

1.1.5 (e) When to consider more principal components:

We would consider additional principal components if the first two components do not capture a sufficient proportion of the total variance for the analysis goals, or if later components reveal important structure or patterns (e.g., clustering) relevant to the research question. Scree plots, cumulative variance thresholds (such as 80%–90%), or domain knowledge about necessary detail can guide this decision.

1.2 Exercise: Data Imputation (60 points)

Data imputation techniques can be used to fill out missing data in sparse dataframes. Here, we will try to generate missing entries in a proteomics dataset ("ms toy.txt") that were below the detection limit for expression data. Missing values are denoted as "NA".

1.2.1 1) Imputation based on a given data distribution (30 points)

-> (a): Write a function that imputes missing values by sampling from a normal distribution whose mean comes from a lower quantile of the observed data and whose standard deviation is a fraction of the observed SD.

1.2.1.1 (a) Implement the imputation function:

The function below takes a numeric vector with NAs, computes the observed mean and SD, sets the imputation mean to `qnorm(percentile, mean, sd)` (lower tail), and the imputation SD to `sd_ratio * sd`. It draws imputed values with `rnorm` and fills the missing entries.

```r
# Imputation based on a given data distribution
impute_from_distribution <- function(x, percentile = 0.05, sd_ratio = 0.5, seed = 42) {

  # --- Step 0: Setup and preprocessing ---

  # set a random seed for reproducibility
  if (!is.null(seed)) set.seed(seed)

  # Separate observed (non-missing) values
  obs <- x[!is.na(x)]

  # --- Step 1: Calculate mean and standard deviation of observed data ---
  m <- mean(obs)
  s <- sd(obs)

  # --- Step 2: Derive new mean from lower quantile of the old distribution ---
  # We use qnorm to get the value at a given lower quantile (e.g., 5%)
  # This simulates the lower tail of the original distribution for imputation
  mu_imp <- qnorm(percentile, mean = m, sd = s)

  # --- Step 3: Derive new standard deviation as a fraction of observed SD ---
  # This ensures the imputed values vary less than the observed data
  sd_imp <- sd_ratio * s

  # --- Step 4: Generate imputed values for missing entries ---
  n_miss <- sum(is.na(x))
  imp_vals <- rnorm(n_miss, mean = mu_imp, sd = sd_imp)

  # Replace missing values with imputed values
  x_imp <- x
  x_imp[is.na(x_imp)] <- imp_vals

  # --- Return results and summary statistics ---
  return(list(
    values = x_imp,       # full vector with imputed values
    imputed = imp_vals,   # the imputed subset
    mu_imp = mu_imp,      # imputation mean
    sd_imp = sd_imp,      # imputation standard deviation
    mean_obs = m,         # observed mean
    sd_obs = s,           # observed standard deviation
    n_missing = n_miss    # number of missing values imputed
  ))
}
```

-> (b): Apply your function to the variable ctrl.1 and plot the overall sample and the imputed data similarly to Figure 1.

1.2.1.2 (b) Apply to ctrl.1 and produce a plot:

We read the dataset, impute `ctrl.1` using a lower-tail mean (`percentile = 0.05`) and a reduced spread (`sd_ratio = 0.5`), and overlay the histograms of the overall values (blue) and imputed values (red).

```r
ms_raw <- read.delim("Assignment_1_supplement/ms_toy.txt", check.names = FALSE)
ctrl1 <- ms_raw$ctrl.1
sum(is.na(ctrl1))  # missing count before
```

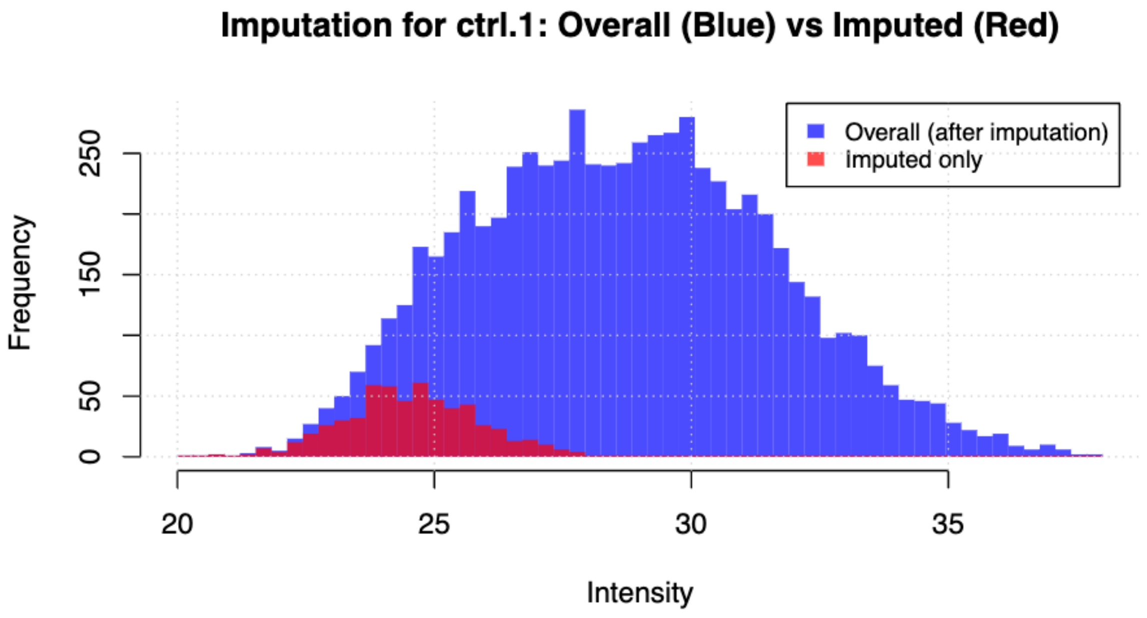
```
## [1] 587
```

```r
imp_ctrl1 <- impute_from_distribution(ctrl1, percentile = 0.05, sd_ratio = 0.5, seed = 123)
sum(is.na(imp_ctrl1$values))  # missing count after
```

```
## [1] 0
```

```r
all_vals <- imp_ctrl1$values
imp_vals <- imp_ctrl1$imputed
brks <- seq(floor(min(all_vals)), ceiling(max(all_vals)), length.out = 60)

hist(all_vals,
     breaks = brks,
     col = rgb(0, 0, 1, 0.7), border = NA,
     main = "Imputation for ctrl.1: Overall (Blue) vs Imputed (Red)",
     xlab = "Intensity")
hist(imp_vals,
     breaks = brks,
     col = rgb(1, 0, 0, 0.7), border = NA,
     add = TRUE)
```

```r
legend("topright", inset = 0.02,
       legend = c("Overall (after imputation)", "Imputed only"),
       fill = c(rgb(0, 0, 1, 0.7), rgb(1, 0, 0, 0.7)), border = NA, cex = 0.8)
grid()
```



```r
data.frame(
  observed_mean = round(imp_ctrl1$mean_obs, 3),
  observed_sd = round(imp_ctrl1$sd_obs, 3),
  imputed_mean = round(imp_ctrl1$mu_imp, 3),
  imputed_sd = round(imp_ctrl1$sd_imp, 3),
  n_missing = imp_ctrl1$n_missing
)
```

```
##   observed_mean observed_sd imputed_mean imputed_sd n_missing
## 1        29.036       2.739       24.531       1.37       587
```

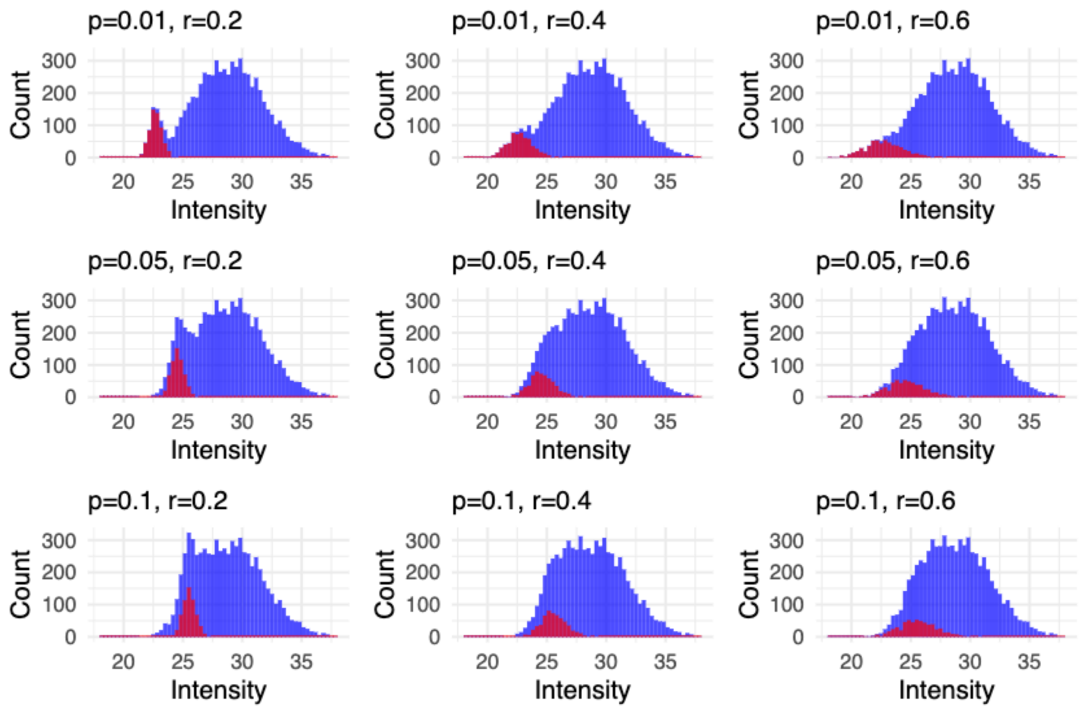
-> (c): What is the effect of changing the percentile parameter in Step 2 and the ratio parameter in Step 3?

1.2.1.3 (c) Parameter effects: Lower percentiles move the imputed mean further into the left tail (smaller values), while larger percentiles shift imputed values upward. Smaller SD ratios produce a tighter imputed distribution (less spread), while larger ratios widen it. The table shows the imputation mean and SD for several settings.

```r
percentiles <- c(0.01, 0.05, 0.10)
ratios <- c(0.2, 0.4, 0.6)
out <- list()
for (p in percentiles) {
  for (r in ratios) {
    tmp <- impute_from_distribution(ctrl1, percentile = p, sd_ratio = r, seed = 123)
    out[[length(out) + 1]] <- data.frame(
      percentile = p,
      sd_ratio = r,
      imputed_mean = tmp$mu_imp,
      imputed_sd = tmp$sd_imp
    )
  }
}
do.call(rbind, out)
```

```
##   percentile sd_ratio imputed_mean imputed_sd
## 1       0.01      0.2     22.66407  0.5478088
## 2       0.01      0.4     22.66407  1.0956176
## 3       0.01      0.6     22.66407  1.6434264
## 4       0.05      0.2     24.53071  0.5478088
## 5       0.05      0.4     24.53071  1.0956176
## 6       0.05      0.6     24.53071  1.6434264
## 7       0.10      0.2     25.52581  0.5478088
## 8       0.10      0.4     25.52581  1.0956176
## 9       0.10      0.6     25.52581  1.6434264
```

Visualization: Histograms showing how the imputed distribution changes with different `percentile` and `sd_ratio` settings.



-> (d): Which configuration of parameters is the most logical/desirable?

1.2.1.4 (d) Recommended configuration:

The goal is to impute data that was “below the detection limit,” which implies the true values are both low and consistent. The parameter p controls the center of the imputed (red) data, and a small p like 0.01 correctly shifts this mean to the low-intensity region, separate from the observed (blue) data. The parameter r controls the spread, and a small r like 0.2 creates a tight, narrow peak. (see the plots above) This narrow peak logically represents a group of values that all failed to cross the same detection threshold. Therefore, combinations like (p=0.01, r=0.2) is one of the combination that creates the distinct, low, and non-variable cluster that we would expect from such missing data.

1.2.2 2) k-Nearest Neighbor imputation (30 points)

-> (a): Create a dataset with only ctrl.1, ctrl.2, ctrl.3; remove rows where all three are missing; then perform kNN imputation.

1.2.2.1 (a) Prepare control-only data and run kNN:

We subset the three control samples and drop rows with no information across all three controls. We then use `VIM::kNN` with `k = 5`.

```r
ms_ctrl <- ms_raw[, c("ctrl.1", "ctrl.2", "ctrl.3")]
all_three_na <- rowSums(is.na(ms_ctrl)) == ncol(ms_ctrl)
ms_ctrl_filt <- ms_ctrl[!all_three_na, ]
```

```r
missing_before_ctrl1 <- sum(is.na(ms_ctrl_filt$ctrl.1))
list(rows_after_filter = nrow(ms_ctrl_filt), missing_ctrl1_before = missing_before_ctrl1)
```

```
## $rows_after_filter
## [1] 6733
##
## $missing_ctrl1_before
## [1] 360
```

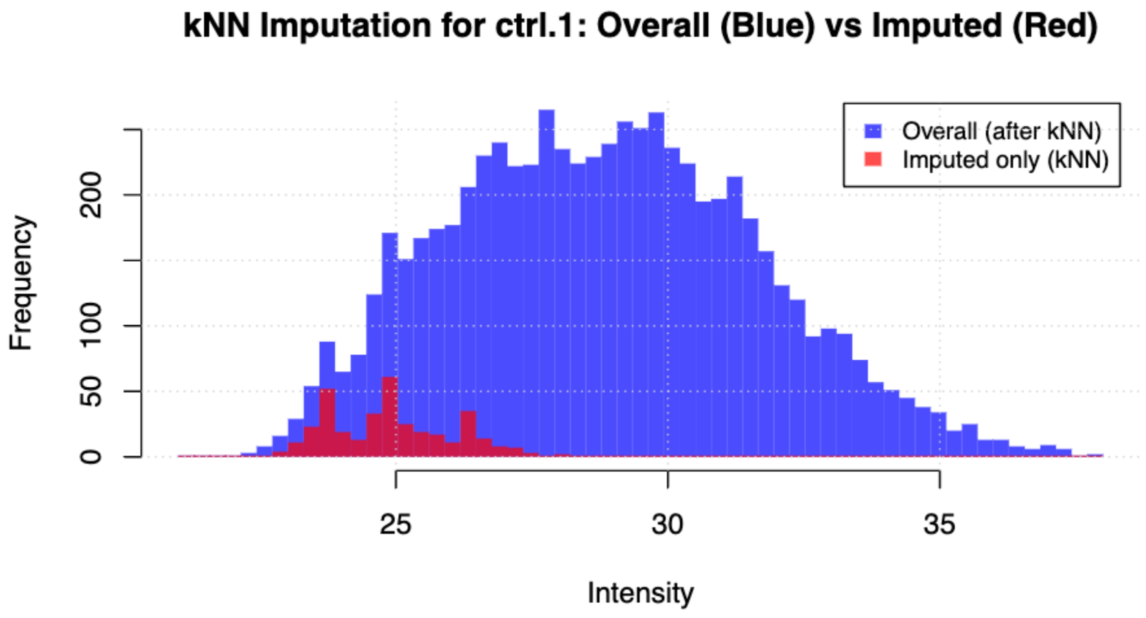
```r
# load VIM
suppressPackageStartupMessages(library(VIM))
knn_res <- VIM::kNN(ms_ctrl_filt, k = 5, imp_var = TRUE)

# Extract imputed indicator and values for ctrl.1
imp_flag_ctrl1 <- knn_res$ctrl.1_imp
if (!is.logical(imp_flag_ctrl1)) imp_flag_ctrl1 <- as.logical(imp_flag_ctrl1)

ctrl1_knn_all <- knn_res$ctrl.1
ctrl1_knn_imp <- ctrl1_knn_all[imp_flag_ctrl1]

list(
  missing_ctrl1_after = sum(is.na(ctrl1_knn_all)),
  imputed_count_ctrl1 = length(ctrl1_knn_imp)
)
```

```
## $missing_ctrl1_after
## [1] 0
##
## $imputed_count_ctrl1
## [1] 360
```

-> (b): Plot the imputed data for ctrl.1 similarly to Figure 1 and compare its distribution.

1.2.2.2 (b) Plot ctrl.1 after kNN imputation: We overlay the histogram of all ctrl.1 values after kNN imputation (blue) with the histogram of imputed-only values (red).

```r
brks2 <- seq(floor(min(ctrl1_knn_all)), ceiling(max(ctrl1_knn_all)), length.out = 60)
hist(ctrl1_knn_all,
     breaks = brks2,
     col = rgb(0, 0, 1, 0.7), border = NA,
     main = "kNN Imputation for ctrl.1: Overall (Blue) vs Imputed (Red)",
     xlab = "Intensity")
hist(ctrl1_knn_imp,
     breaks = brks2,
      col = rgb(1, 0, 0, 0.7), border = NA,
      add = TRUE)
legend("topright", inset = 0.02,
       legend = c("Overall (after kNN)", "Imputed only (kNN)"),
       fill = c(rgb(0, 0, 1, 0.7), rgb(1, 0, 0, 0.7)), border = NA, cex = 0.8)
grid()
```



-> (c): Compare the kNN result to the distribution-based imputation in terms of mean, SD, number of values (overall and imputed-only).

1.2.2.3 (c) Quantitative comparison: The table contrasts summary statistics for ctrl.1 using the two methods. Note: kNN uses the filtered dataset (rows where not all three controls are missing), while the distribution-based method was applied to the full dataset.

```r
compare_df <- rbind(
  data.frame(
    method = "Distribution",
    n = length(imp_ctrl1$values),
    missing_before = sum(is.na(ms_raw$ctrl.1)),
    imputed_n = length(imp_ctrl1$imputed),
    mean_overall = mean(imp_ctrl1$values),
    sd_overall = sd(imp_ctrl1$values),
    mean_imputed = mean(imp_ctrl1$imputed),
    sd_imputed = sd(imp_ctrl1$imputed)
  ),
  data.frame(
    method = "kNN",
    n = length(ctrl1_knn_all),
    missing_before = missing_before_ctrl1,
    imputed_n = length(ctrl1_knn_imp),
    mean_overall = mean(ctrl1_knn_all),
    sd_overall = sd(ctrl1_knn_all),
    mean_imputed = mean(ctrl1_knn_imp),
    sd_imputed = sd(ctrl1_knn_imp)
  )
)
compare_df
```

```
##         method    n missing_before imputed_n mean_overall sd_overall
## 1 Distribution 6960            587       587     28.65922   2.925119
## 2          kNN 6733            360       360     28.81819   2.832218
##   mean_imputed sd_imputed
## 1     24.56816   1.312180
## 2     24.96160   1.224823
```

-> (d): What are the advantages and disadvantages of kNN imputation compared to the distribution-based approach?

1.2.2.4 (d) Discussion:

- **kNN advantages**: leverages multivariate structure (uses similarity across ctrl.1–3), preserves local relationships and can adapt to non-Gaussian patterns; may yield more realistic values for correlated variables.
- **kNN disadvantages**: requires tuning `k`; sensitive to scaling and outliers; can bias towards dense regions; depends on available neighbors (after filtering) and may be computationally heavier on large datasets.
- **Distribution-based advantages**: simple, fast, controlled placement of imputed values in the lower tail to reflect left-censoring; easy to explain and reproduce.
- **Distribution-based disadvantages**: ignores relationships between variables; assumes normality and chosen tail/variance; risk of under/over-dispersion if parameters are poorly chosen.